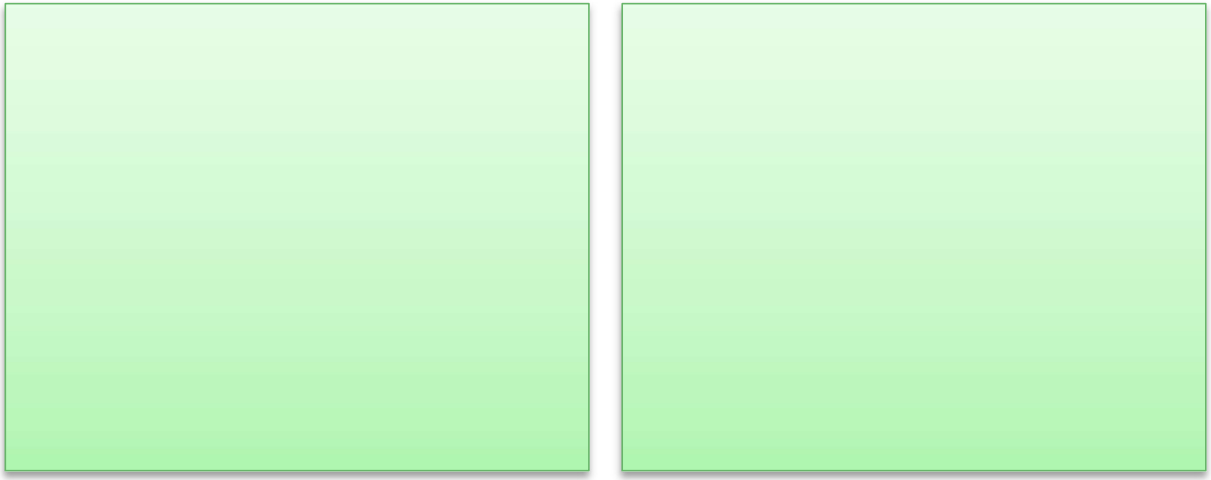
Identitet och likhet

Identitet och likhet

- Är **identitet** och **likhet** samma sak?

> **– Oj, vi har samma kläder på oss idag!**
> **– Nej, men likadana!**
>
> *– Besserwisser*

  - **Två rutor** som:
    - Är **exakt likadana**
    - Men inte **samma** – de har **varsin identitet**



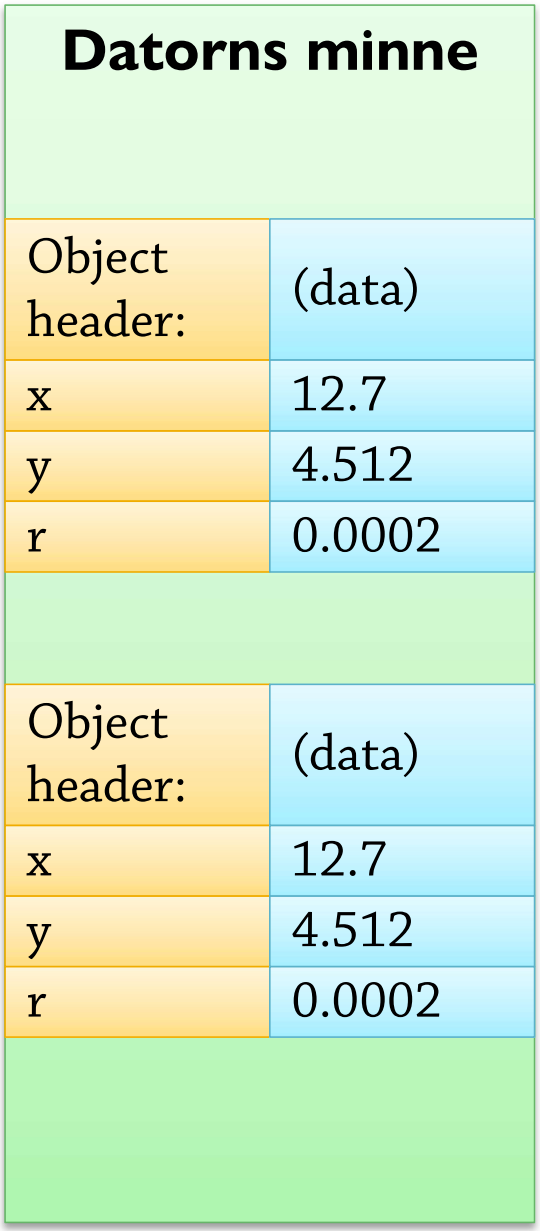
Identitet i Java

(vad är "samma" objekt?)

Identitet

- "Samma objekt" **betyder** "samma adress"
  - **Ser** jag *två saker* i minnet,
    så är de inte **samma** sak:
    Då vore de ju *en sak*

- Olika adress = olika identitet!
  - Även om objektens **tillstånd, ”state”**
    (informationen lagrad i objekten)
    råkar vara lika
  - Två *distinkta lådor*
    kan ha *identiskt innehåll*



Identitet och ==

== jämför värdet på två variabler

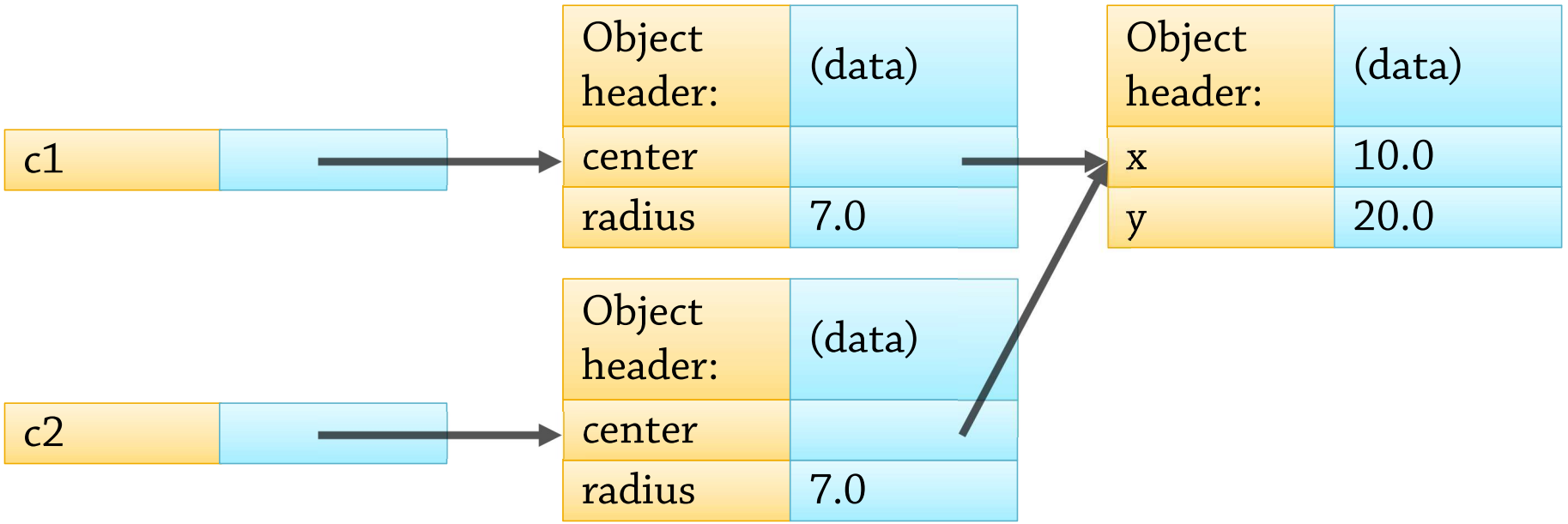
Värdet av en objektvariabel är en pekare (adress)

== jämför pekare

== testar om två pekare pekar på samma objekt!

c1 != c2
c1.center == c2.center

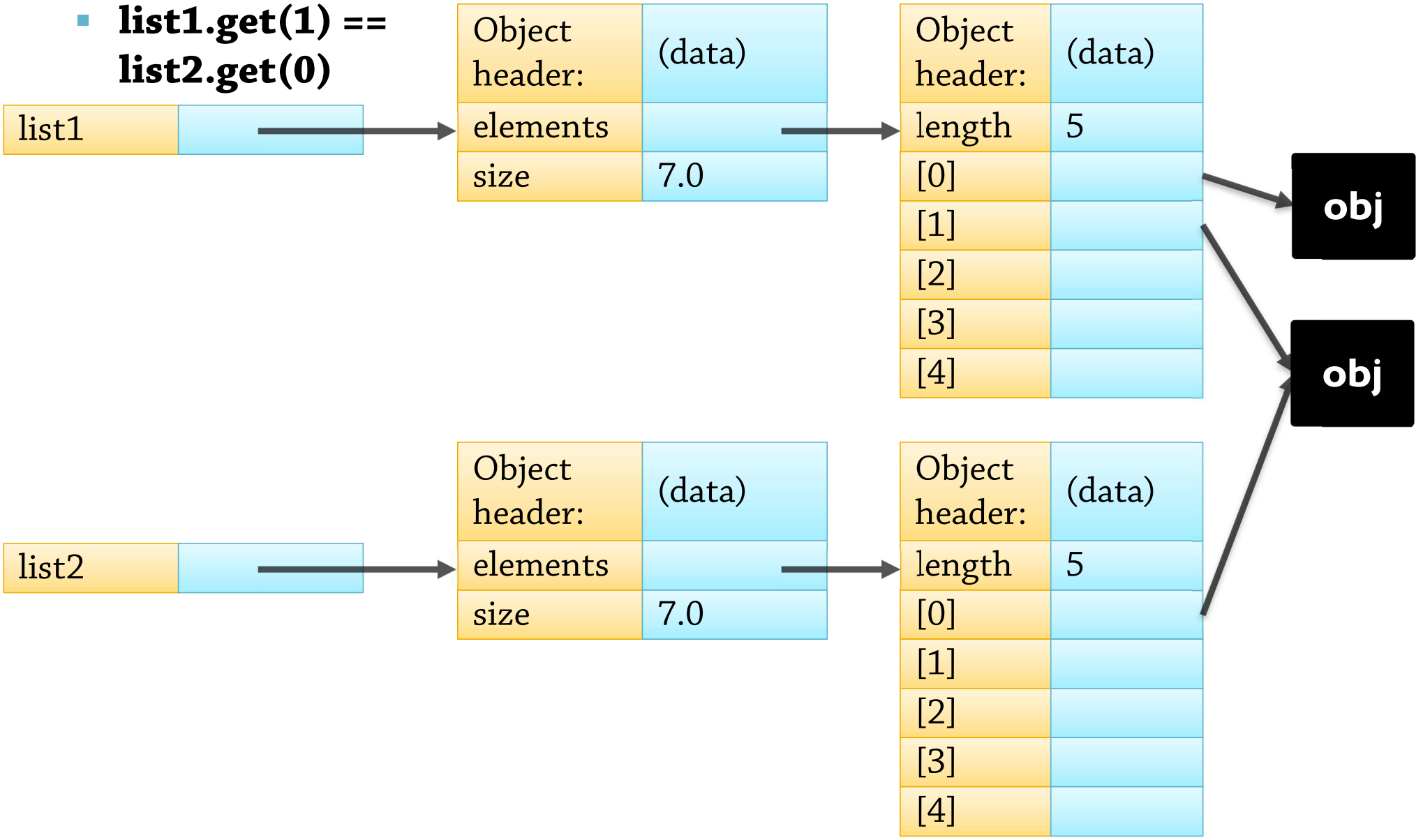
Python: c1 is not c2
c1.center is c2.center



Samma objekt i listor

jonkv@ida

- Kan även ha två **listor** som pekar ut samma objekt
  - **list1.get(1) == list2.get(0)**



Samma objekt i listor (2)

- Varför vill man det?
  - Exempel: Filtrera en lista
  - Resultatet pekar ut *samma personobjekt* som i ursprungslistan
  - Behöver inte *lagra alla personobjekt dubbelt*

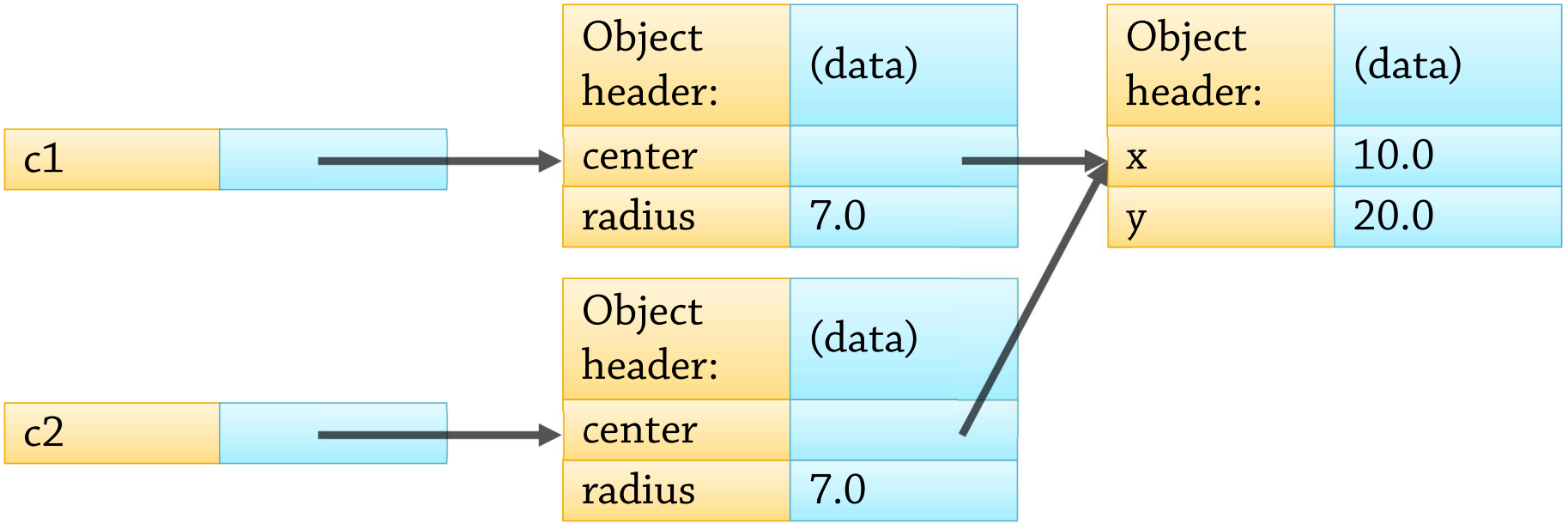
```
public class Filter {
   public static List<Person> findAdults(ArrayList<Person> persons) {
      List<Person> result = new ArrayList<>();
      for (Person person : persons) {
         if (person.isAdult()) {
            result.add(person);
         }
      }
      return result;
   }
}
```

Identitet för primitiva värden?

Ingen identitet

- Primitiva värden **har ingen identitet**
  - Kan inte peka på dem (bara på objekt som innehåller dem)
  - ➔ Meningslöst att försöka skilja på ”samma sjua” och ”två olika men precis likadana sjuor”

- **c1.radius == c2.radius**
  - Jämför inte *pekare / identitet* utan *värdet, som är själva flyttalet*

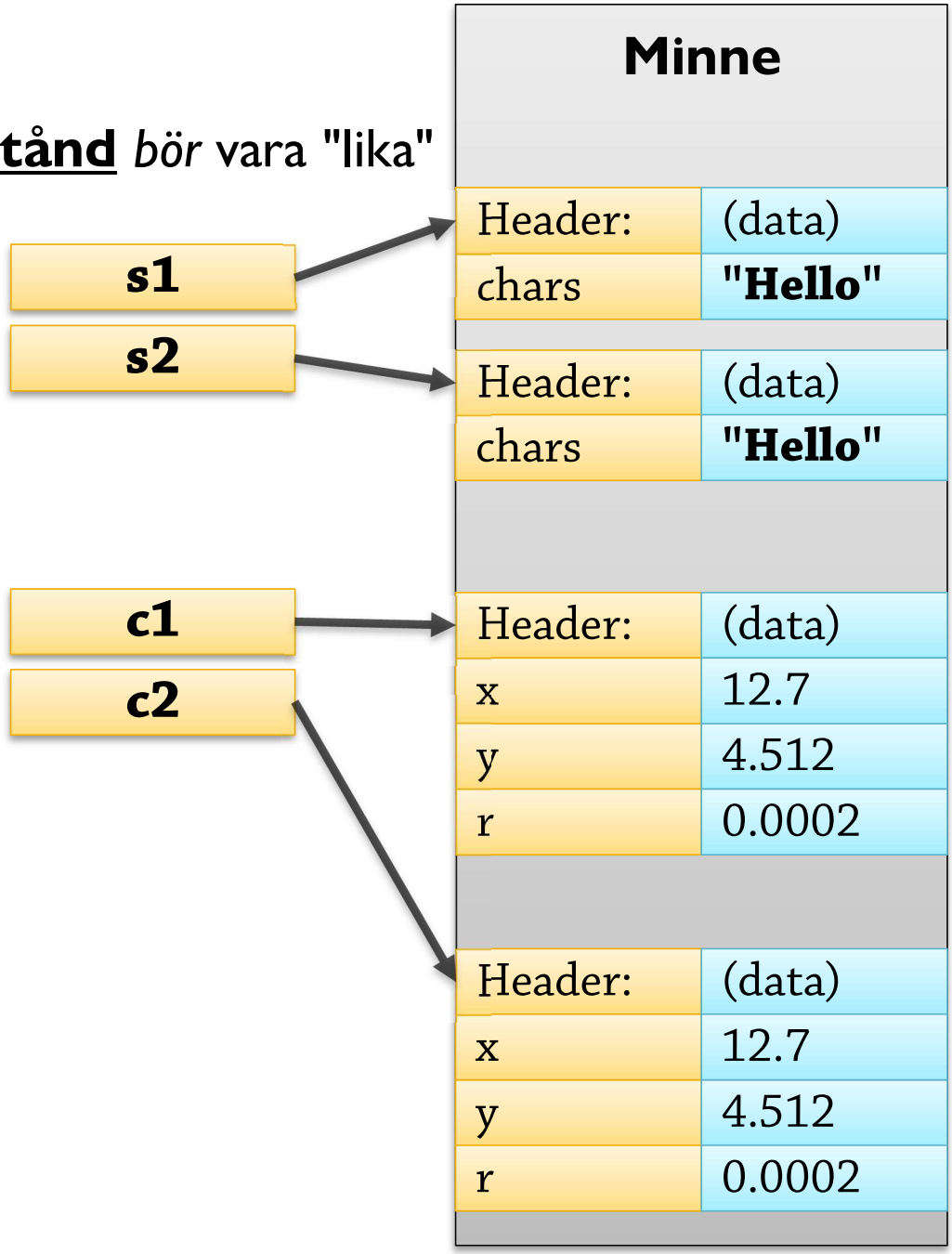


Likhet i Java

Likhet = identiskt tillstånd?

jonkv@ida

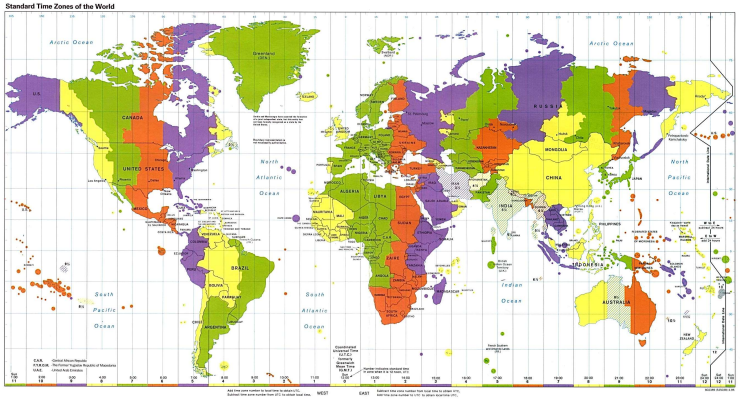
- Vad **betyder** likhet?
  - 1) Två objekt med **identiskt tillstånd** *bör* vara "lika"
    - Exempel: Två strängobjekt på olika plats i minnet **innehåller samma tecken**
    - **s1 != s2**, men vi tycker objekten är "lika"
    - Exempel: Två cirkelobjekt med samma position och radie
    - **c1 != c2**, men vi tycker objekten är "lika"



Likhet: Även vid olika tillstånd!

- Scenario: Vi vill hitta fel i cirkelklassen
  - Lägger till **tidsstämpel** för när varje objekt skapades
  - Objekt med **olika tidsstämpel** kan ändå anses "**lika**"!

- Scenario: Vi håller reda på datum och tider
  - Fält för datum, klockslag och **tidszon**
  - Vi vill (kanske) att
    12:00 GMT är "samma tid" som
    13:00 WET (Western European Time)



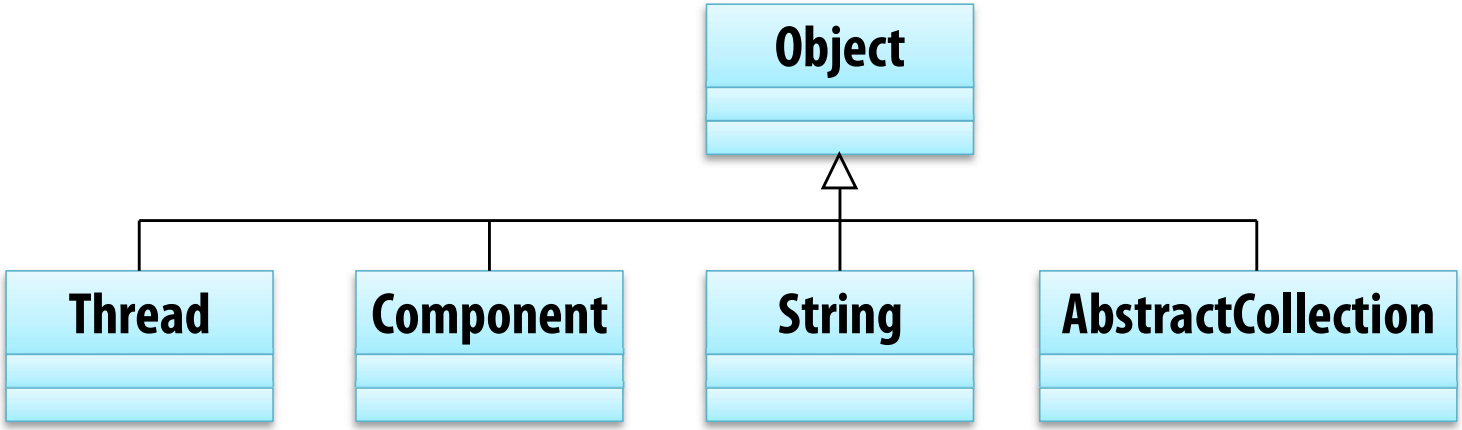
Datorns minne

Object header:	(data)
x	12.7
y	4.512
r	0.0002
created	4711.31
Object header:	(data)
x	12.7
y	4.512
r	0.0002
created	4715.77

➔ **"Likhet" är inte samma sak som "alla fält har samma värde"**

Likhet: Inte fördefinierad

- **Vi** måste definiera likhet – vad menar vi egentligen?
  - Ska *finnas* för alla klasser
    - Klassen **Object** definierar metoden **equals**(), ärvs till alla andra klasser



  - Kan behöva vara *olika* för olika klasser
    - Använd overriding för att definiera om **equals**()

  - **Används** av *många* klasser och metoder
    - List.contains(element) tittar om listan har ett element som är *lika / equal*
    - List.indexOf(element) ger positionen för ett element som är *lika / equal*

Likhet: Metodsignatur

- Måste kunna **jämföra** objekt **utan att veta typen i förväg**!
  - Lösning: equals() tar Object som parameter
    - ➔ Accepterar vilken typ av objekt som helst

Object.java

```
public class Object {
   public boolean equals(Object other) {
      ...
   }
}
```

MyList.java

```
public class MyList {
   public boolean contains(Object other) {
      for (int i = 0; i < size(); i++) {
         Object atThisPos = this.getElementAt(i);
         if (atThisPos.equals(other)) return true;
      }
      return false;
   }
}
```

equals() *finns* i alla objekt – även atThisPos
equals() *accepterar* alla objekt – även other

Likhet: Standardbeteende

- Måste ha en **första implementation** i **Object**
  - Men **Object** har inga fält
  - Enda alternativen:
    - Två **Object** anses alltid **lika** (per default)?
    - Två **Object** anses alltid **olika** (per default)?
  - Vill inte **riskera** att returnera sant om objekten borde ses som olika ➔

Object.java

```
public class Object {
   public boolean equals(Object other) {
      return this == other;
   }
}
```

Likhet: Standardbeteende

- I andra klasser: Overriding
  - Åtminstone *om* man behöver jämföra objekt av denna typ!

Circle.java

```
public class Circle {
   // Cirklar jämför man så här!
   public boolean equals(Object other) { … }
}
```

Parametern måste vara av typ Object, inte Circle: Vi måste kunna jämföra en cirkel med vad som helst!

Likhet: Några krav

jonkv@ida

- Några **krav** på likhet (en *ekvivalensrelation*):
  - Måste vara **reflexiv**: x.equals(x)
  - Måste vara **symmetrisk**: x.equals(y) ➔ y.equals(x)
  - Måste vara **transitiv**: x equals(y) och y.equals(z) ➔ x.equals(z)
  - Får inte returnera sant för null: x.equals(null) == **false**

Ger ramar att hålla oss till

Implementera equals(): Circle

jonkv@ida

- En equals() för Circle:

```
public class Circle {
   protected double x, y, r;
   public boolean equals(final Object o) {
      if (this == o) return true;      // Alltid lika med sig själv
      if (o == null) return false;     // Aldrig lika med null
      if (getClass() != o.getClass()) return false; // Måste vara samma klass

      final Circle that = (Circle) o;

      if (this.x != that.x) return false;
      if (this.y != that.y) return false;
      if (this.r != that.r) return false;

      return true;
   }
}
```

OK, det var en cirkel:
Se till att få rätt pekartyp

Testar våra egna krav:
Samma x, y och radie

Allt OK: Är lika

Implementera equals(): Objektfält

jonkv@ida

- När vissa **fält** är **objektpekare**: Hur jämför vi?

```
public class ColorCircle {
   protected double x, y, r;
   protected Color color;
   public boolean equals(final Object o) {
      if (this == o) return true;
      if (o == null || getClass() != o.getClass()) return false;

      final ColorCircle that = (ColorCircle) o;
      if (this.x != that.x) return false;
      if (this.y != that.y) return false;
      if (this.r != that.r) return false;
      if (this.color != that.color) return false;

      return true;
   }
}
```

!= testar om that.color pekar på samma färgobjekt

```
Circle gc1 = new ColorCircle(10, 11, 12, new Color(0,0,0));
Circle gc2 = new ColorCircle(10, 11, 12, new Color(0,0,0));
```

gc1.equals(gc2) är falskt!

Implementera equals(): Objektfält

jonkv@ida

- Använd equals() för **objektfält**!

```
public class ColorCircle {
   protected double x, y, r;
   protected Color color;

   public boolean equals(final Object o) {
      if (this == o) return true;
      if (o == null || getClass() != o.getClass()) return false;

      final ColorCircle that = (ColorCircle) o;

      if (this.x != that.x) return false;
      if (this.y != that.y) return false;
      if (this.r != that.r) return false;
      if (!this.color.equals(that.color)) return false;

      return true;
   }
}
```

equals() testar om det är samma färgnyans

Implementera equals(): Objektfält, null

jonkv@ida

- När vissa **fält** kan vara **null**:

```
public class ColorCircle {
   protected double x, y, r;
   protected Color color;      // Kan vara null

   @Override public boolean equals(final Object o) {
      if (this == o) return true;
      if (o == null || getClass() != o.getClass()) return false;

      final ColorCircle that = (ColorCircle) o;
      if (this.x != that.x) return false;
      if (this.y != that.y) return false;
      if (this.r != that.r) return false;
      if (this.color != null && !this.color.equals(that.color)) return false;
      if (this.color == null && that.color != null) return false;

      return true;
   }
}
```

Har vi en färg? Kolla att den är lika med det andra objektets.

Saknar vi färg? Kolla att den andra också saknar det.

Implementera equals(): Ärvning

jonkv@ida

- När vi behöver låta **superklassen** testa sin likhet:

```
public class Circle {
   protected double x, y, r;
   public boolean equals(Object other) {
      if (this == o) return true;          // Alltid lika med sig själv
      if (o == null) return false;         // Aldrig lika med null
      if (this.getClass() != o.getClass()) return false; // Måste vara samma klass
      … testa x, y, r …
   }
}
```

```
public class GraphicCircle extends Circle {
   protected Color outline, fill;

   public boolean equals(final Object o) {
      if (!super.equals(o)) return false;

      final GraphicCircle that = (GraphicCircle) o;
      if (!outline.equals(that.outline)) return false;
      if (!fill.equals(that.fill)) return false;
      return true;
   }
}
```

Superklassen testar:
null, klass, superklassens fält
Vi testar: Våra nya fält

Samma tillstånd

- Två **distinkta objekt** i minnet *kan* alltså ha **samma tillstånd**
  - Precis som två distinkta variabler kan ha samma värde
    - **x=10**
      y=10
    - **x == y**, men x och y är ändå olika variabler (två lådor, identiskt innehåll)!

Python:
a = ["hello", "world”]
b = ["hello", "world”]
➔ **två** listor med **samma** element
➔ **(a == b)** and **(a is not b)**

b.append("again")
➔ bara b ändras, inte a
➔ a och b var *två listor* på *varsin plats* i minnet

Datorns minne

Object header:	(data)
x	12.7
y	4.512
radius	0.0002

Object header:	(data)
x	12.7
y	4.512
radius	0.0002

Samma tillstånd, men inte samma objekt